VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Radim Budáč

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Radim Budáč**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. RNDr. Petr Šaloun, Ph.D.**

Konzultant bakalářské práce: Ing. Roman Rotter

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 23. dubna 2019 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

Tieto Czech s.r.o.
28. října 3346/91
702 00 Ostrava - Moravská Ostrava
IČO 64608441 DIČ CZ64608441

V Ostravě 23. dubna 2019

.....

Rád bych zde poděkoval svému vedoucímu práce, panu doc. RNDr. Petru Šalounovi, Ph.D. za vedení a věcné rady. Dále bych rád poděkoval svému konzultantovi panu Ing. Romanu Rotterovi za ochotu, podporu a pomoc během absolvování praxe. V poslední řadě chci poděkovat všem svým kolegům, se kterými jsem měl možnost pracovat.

Abstrakt

Tato bakalářská práce popisuje individuální odbornou praxi ve firmě Tieto Czech s.r.o. První kapitola, znázorněná krátkým úvodem, je následována popisem firmy spolu s pracovním zařazením studenta. Další kapitola popisuje průběh praxe spolu s využitým softwarem a technologiemi, které byly během praxe využity. V další části jsou uvedeny a krátce popsány úkoly, které musel student řešit, následovány kapitolou s detailnějším popisem jejich řešením. Předposlední kapitola je věnována časovému rozvržení během praxe. Poslední kapitola práce popisuje, které znalosti studenta byly v praxi používány, které mu chyběly a které získal v praxi. Kapitola je zakončena celkovým zhodnocením praxe.

Klíčová slova: Tieto, MVVM, WPF komponenta, ASP.NET Core, MVC, MS SQL, XAML, Repozitář

Abstract

This bachelor thesis describes individual professional practice in company Tieto Czech s.r.o. The first chapter, illustrated by short introduction, is followed by the company description along with the student's grade. The next chapter describes the practice along with the software and technologies used. In the next part are presented and briefly described tasks, which the student had to solve, followed by a chapter with a more detailed description of their solution. The penultimate chapter is devoted to time layout during practice. The last chapter of the thesis describes which of the student's knowledge were used in practice, which he lacked and which he acquired in practice. The chapter is completed by an overall assessment of practice.

Key Words: Tieto, MVVM, WPF component, ASP.NET Core, MVC, MS SQL, XAML, Repository pattern

Obsah

Seznam použitých zkratek a symbolů

WPF	–	Windows Presentation Foundation
MVVM	–	Model-View-ViewModel
MVC	–	Model-View-Controller
MS SQL	–	Microsoft SQL Server
XAML	–	Extensible Application Markup Language
SOC	–	Security Operations Center
XML	–	Extensible Markup Language
AD	–	Active Directory
API	–	Application Programming Interface
TFS	–	Team Foundation Server
TFVC	–	Team Foundation Version Control
HTML	–	Hyper Text Markup Language
SQL	–	Structured Query Language
GUI	–	Graphical User Interface
SDK	–	Software Development Kit
ORM	–	Object-relational Mapping
WCF	–	Windows Communication Foundation
EPS	–	Event Per Second

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Příležitost pracovat pro firmu Tieto Czech s.r.o. na pozici software developer se mi naskytla již na konci zimního semestru v prvním ročníku. Z tohoto důvodu jsem se rozhodl, že svou závěrečnou bakalářskou práci vykonám formou absolvování individuální odborné praxe, kterou firma nabízí.

Nejen proto, že už ve firmě nějakou dobu pracuji, ale také protože mě práce programátora baví a chtěl bych se jí dál věnovat. Navíc si myslím, že pokračování v získávání pracovních zkušeností je důležité do budoucna.

V této práci zprvu krátce popisuji firmu, ve které pracuji, její historii a oblast zaměření. Dále popisuji své pracovní zařazení, v jakém týmu a na jaké pozici jsem se během praxe nacházel. Následuje stručný popis průběhu praxe, a poté technologie a software, které jsem při své práci používal.

Jako další se zaměřuji na zadání úkolů, kterým jsem se věnoval. Následuje detailnější popis postupu řešení těchto úkolu, který je příležitostně obohacený o části zajímavého kódu.

Předposlední kapitola je věnována časovému rozvržení během praxe. Tedy kolik času jsem strávil programováním, učením nových technologií a podobně. Dále je v kapitole uvedená časová náročnost jednotlivých úkolů.

Poslední kapitola práce je věnována znalostem, které jsem během své praxe uplatnil, ale také těm které mi scházely a postupně jsem je nabyl. Práce je zakončena celkovým zhodnocením absolvované odborné praxe.

2 Popis odborného zaměření firmy a mého pracovního zařazení

2.1 Tieto

Tieto [1] je společnost, která byla založena roku 1968 ve finském městě Espoo pod názvem Tietotehdas Oy. Původní zaměření společnosti bylo poskytování IT služeb pro banky a lesní průmysl ve Finsku. V 80. letech se zaměření rozšířilo na služby pro osobní počítače a vývoj softwaru. 90. léta byla pro společnost znamením velkého a rychlého růstu prostřednictvím uzavírání nových kontraktů a strategických aliancí. Název společnostit se roku 1995 změnil na TT Tieto a o 3 roky později na Tieto. Roku 1999 se Tieto spojilo se společností Enator a vzniklo TietoEnator. Díky rychlé globalizace IT průmyslu v roce 2000 se společnosti dařilo urychlit vývoj na mezinárodních trzích. Nynější název, tedy Tieto, nese společnost od roku 2009.

Společnost Tieto [2] má přes 15 000 zaměstnanců v téměř 20 zemích, z toho přes 2600 zaměstnanců právě v České Republice do které se dostala v roce 2001. V Ostravě bylo roku 2004 otevřeno softwarové centrum. O 8 let později bylo v Ostravě postaveno nové sídlo Tieto Towers[3], do kterého se přesídlili zaměstnanci ze čtyř budov. Nyní je sídlo Tieto Towers třetí největší pobočkou celé společnosti. Tato pobočka poskytuje vývoj softwaru, poradenství a řešení v problematice IT, správu infrastruktury a kompletní IT řešení pro významné zákazníky společnosti jako jsou finanční a telekomunikační společnosti.

2.2 Pracovní zařazení

Ve firmě jsem byl zařazen do nově vznikajícího oddělení Security Operations Center (SOC) na pozici Junior Software Developer. V době mého nástupu disponoval vývojářský tým pouze čtyřmi programátory a to včetně mě a mého kolegy ze školy. Vzhledem k množství implementační práce, potřebné pro chod SOC oddělení, se vývojářský tým rozrostl na požadovaných deset zaměstnanců. Tato organizační změna, spolu s množstvím projektů, vyžadovala přechod na řízení produktového vývoje pomocí agilní metodiky Scrum[1].

[1] Empirický proces, který pomocí inkrementální a iterativní formy dodává zákazníkovy hodnoty v krátkých cyklech tak, aby byla získána pravidelná a častá zpětná vazba.

3 Průběh praxe

Jelikož jsem ve firmě pracoval už před nástupem na praxi, byl jsem ušetřen procesu, které musí nový zaměstnanec absolvovat, jako je vstupní školení, seznámení s prostředí firmy a seznámení s členy týmu, ve kterém bude pracovat.

Z důvodu, že jsem byl vedený jako stážista, nevztahovala se na mě fixní pracovní doba a mohl jsem pracovat kdykoli. Samozřejmě bylo dobré být v práci zároveň s kolegy kvůli schůzím a případným konzultacím problémů. Každý zaměstnanec SOC oddělení byl povinen si zapisovat budoucí směny do aplikace Humanity (viz kapitola 3.1.4), to se týkalo i mě.

Průběh běžného pracovního dne začínal ranním příchodem do práce, kdy jsem měl zhruba dvě hodiny na programování před tím, než začala denní Scrum schůze. Hlavním cílem těchto schůzí bylo identifikovat a odstranit překážky, které nastaly při práci na dané úloze, kromě toho také zhodnotit, co programátor udělal za poslední den a čemu se bude v daný den věnovat. Následovalo opět programování, případně konzultace mých úkolů nebo problémů, se kterými jsem se setkal.

Jednou za čtrnáct dní probíhala schůze na plánování Sprintu. Na této schůzi se prostřednictvím webové aplikace Jira přiřazovaly úkoly programátorům na budoucí dva týdny. Každý úkol měl svou prioritu, předpokládanou časovou náročnost a stručný popis. Vždy před zahájením práce na jednotlivých úkolech jsem změnil jejich status v aplikaci Jira na "In Progress". Každý úkol měl i své jedinečné označení, které se využívalo jako název Git větve. Po dokončení práce jsem lokální větev nahrál na TFS a vytvořil požadavek na její sjednocení s hlavní větví. Do těchto požadavků jsem vždy přidal svého nadřízeného, v případě potřeby i další kolegy, na revizi kódu. Po schválení došlo ke sjednocení, sestavení a následnému nahrání nové verze aplikace na testovací prostředí. Zároveň jsem úkolu nastavil status "Done" a práci na něm ukončil.

Během absolvování praxe jsem se také občasně účastnil online schůzek SOC oddělení a různých školení, například jak pracovat se systémem Git. V neposlední řadě jsem se příležitostně účastnil revize kódů mých kolegů.

3.1 Použitý software

3.1.1 Visual Studio

Jako software developer jsem při programování používal vývojové prostředí Visual Studio 2017[4].

3.1.2 SQL Server Management Studio

Pro přístup a práci s databází, jako je vytváření tabulek a SQL dotazů, jsem požíval SQL Server Management Studio[5]. Později přešlo oddělení na SQL Server Database Project ve Visual Studiu, který umožňoval verzování pomocí systému Git.

3.1.3 Jira

Na řízení produktového vývoje používá oddělení webovou aplikaci Jira[6], která se používá u agilních týmů. Umožňuje plánování sprintů, distribuování úkolů, prioritizaci a diskuse. Dále se využívá na vytváření reportů, roadmap a správu projektů. Oddělení však používá tuto aplikaci pouze na správu úkolů a plánování sprintů, pro správu roadmap a projektů využívá OpenProject.

3.1.4 Humanity

Humanity[7] je aplikace, která slouží pro správu docházky zaměstnanců. Umožňuje rozdělovat zaměstnance do týmů, snadno vytvářet směny, přiřazovat je zaměstnancům a směňovat je. Dále nabízí možnost nastavit hodiny dostupnosti, aby nedocházelo ke konfliktům při přiřazování směn. V neposlední řadě umožňuje upozornit zaměstnance na jakékoli změny pomocí emailu. Humanity je také systém na vytváření reportů, například pro zobrazení nákladů společnosti na pracovní sílu za určité období.

3.1.5 Team Foundation Server

Na správu verzí, životního cyklu aplikací a interních knihoven používá oddělení TFS[8], který umožňuje automatizované sestavení kódu, spuštění Unit testů, začlenění revize kódu a jiných podmínek do schvalovacího procesu a následně nasazení aplikace na testovací nebo produkční prostředí. TFS nabízí dva způsoby, jakými bude projekt verzován, a to Git nebo TFVC. Zprvu se používal systém TFVC, ale s rozrůstajícím se týmem přešlo oddělení postupně na systém Git s izolovanými větvemi na každý funkční požadavek nebo opravu chyb.

3.2 Použité technologie

3.2.1 WPF

WPF je GUI framework, který je součástí .NET Frameworku od verze 3.0, sloužící pro vytváření desktopových aplikací. WPF je nástupce Windows Forms, který nabízí nové funkce a používá jazyk XAML[2] pro definici uživatelského rozhraní. WPF aplikace jsou nativně renderované pomocí DirectX a při použití DirectX ve verzi 7.0 a výše využívají hardwarovou akceleraci. WPF dále umožňuje používání stylů a hlavně obousměrný binding, který dovoluje užití vzoru MVVM.

3.2.2 MVVM

Model-View-ViewModel je architektonický vzor, který rozděluje aplikaci na tři horizontální vrstvy. Model, reprezentující objekty, které zapouzdřují data a chování aplikační domény. View, které definuje rozvržení uživatelského rozhraní, styly a triggery, a nemělo by mít vliv na funkčnost aplikace. ViewModel, což je třída, která uchovává stav view a má metody implementující

[2]Deklarativní značkovací jazyk zjednodušující vytváření GUI elementů aplikacím, které využívají .NET Framework, je to varianta XML od společnosti Microsoft.

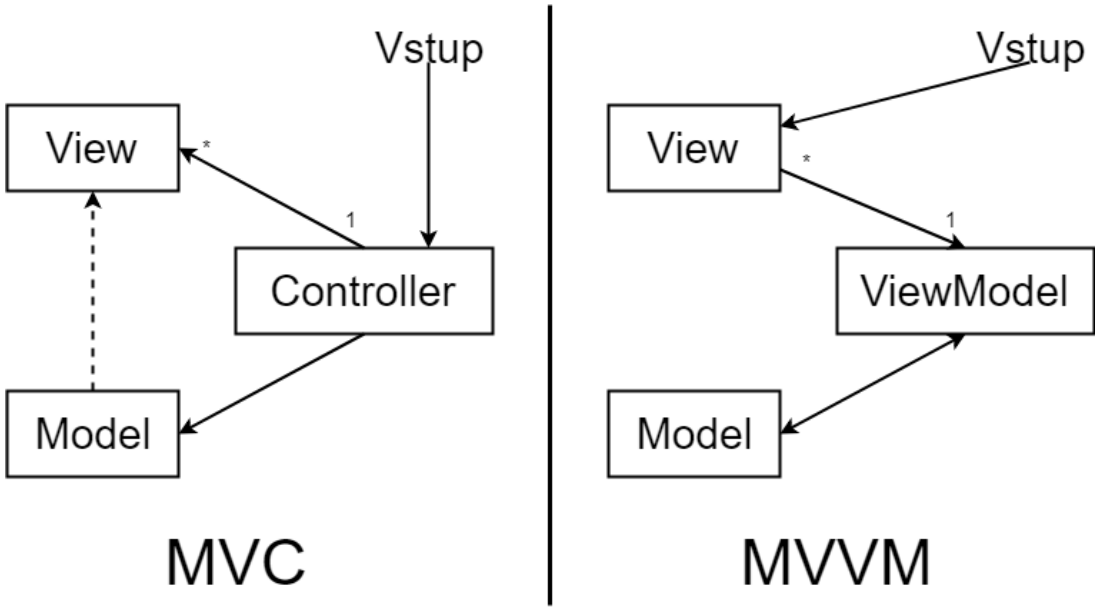
logiku aplikace. Cílem MVVM je zejména testovatelnost aplikace, uživatelské vstupy jdou snadno nahradit unit testy a také umožňuje oddělit programátorskou část od definice UI, která se dá upravovat v programu MS Blend.

3.2.3 MVC

Architektonický vzor, který se využívá zejména na webu, jehož základní myšlenkou je oddělení logiky od výstupu. MVC aplikace je rozdělena na komponenty třech typů, Model, View a Controller. Model obsahuje logiku aplikace, například výpočty, databázové dotazy a validaci. View slouží pro zobrazování výstupu uživateli, nejčastěji se jedná o šablonu obsahující HTML stránku a štítky umožňující vkládat proměnné. Poslední typ controller je prostředník, se kterým komunikuje uživatel, view i model. Obsluhuje požadavky uživatele a propojuje view s modelem. Rozdíl oproti MVVM je, že střední vrstva (controller) je bezestavová, další rozdíly (viz tabulka 1 a obrázek 1).

Tabulka 1: Rozdíl mezi MVC a MVVM

MVC	**MVVM**
Controller je vstupním bod aplikace	View je vstupním bodem aplikace
View nemá odkaz na Controller	View má odkaz na ViewModel
View má vědomí o Modelu	View nemá vědomí o Modelu



Obrázek 1: Rozdíl mezi MVC a MVVM

3.2.4 ASP.NET Core

ASP.NET Core[9] je multiplatformní, vysoce výkonný, open-source framework, který slouží pro vytváření webových aplikací a služeb.

3.2.5 Repozitář (Repository Pattern)

Repozitář je abstrakce datové vrstvy. Vzor odděluje logiku přístupu k datům a mapuje ji na entity business logiky. V repozitáři spolu entity domény, logika přístupu k datům a business logika komunikují pomocí rozhraní.

Repozitář skýtá různé výhody, jako je snadnější údržba kódu díky centralizaci logiky přístupu k datům. Business logiku i logiku přístupu k datům lze testovat samostatně a nevzniká žádná duplicita kódu.

4 Zadání úkolů během odborné praxe

4.1 XML Editor konfiguračních souborů

Prvním úkolem, který mi byl zadán, bylo vytvořit počítačovou aplikaci, která měla sloužit k editaci XML souborů používaných pro konfiguraci Windows služeb, určených pro přenos dat. Aplikace měla umět automaticky načíst výchozí soubor z cílového adresáře, pokud existuje, a tento soubor naformátovat, tedy odřádkovat a odsadit jednotlivé XML tagy (může se stát, že soubor bude v minimalizovaném stavu) aby bylo snazší se v něm orientovat a editovat ho. Další funkcí aplikace mělo být jednoduché šifrování a dešifrování textu, například connection string v XML atributu.

K dispozici mi byla starší verze aplikace, která byla vytvořena technikou code behind, čehož jsem se měl vyvarovat a použít vzor MVVM.

4.2 Cron Maker

Druhým úkolem bylo vytvoření znovupoužitelné WPF komponenty, umožňující uživatelsky vygenerovat CRON řetězec, používaný například pro plánování úloh pomocí Quartz.NET[3].

Předlohou pro uživatelské rozhraní byla oblíbená veřejně dostupná webová verze[10] obdobného nástroje, kterou do té doby používali kolegové ze Security týmu. Cílem byla nativní desktopová verze, včetně oprav nedostatků veřejně dostupného generátoru.

Obrázek 2: Předloha interního generátoru

4.3 SOC Portál

Posledním, nejdůležitějším úkolem, byla práce na SOC Portálu. SOC Portál je víceúčelová interní webová aplikace, která je vyvíjena pomocí ASP.NET Core frameworku s použitím MVC architektury.

Prvním dílčím úkolem na tomto projektu bylo samotné založení projektu, automatické přihlášení uživatele do portálu, implementace základní kostry stránky (tj. kontejneru, navigačního

[3]Open-source systém na plánování a spouštění Quartz Jobů (jakákoli třída, která implementuje jednoduché IJob rozhraní)[11].

menu a společných prvků) podle návrhu, který vytvářel kolega a integrace s AD pro získání dalších informací o přihlášeném uživateli.

4.3.1 Hlášení incidentů

Primární funkcí aplikace bylo jednotné rozhraní, umožňující nahlášení bezpečnostního incidentu kýmkoliv v rámci intranetu (potencionálně 20 tisíc uživatelů), přičemž výstupní akce mohla být konfigurovatelná, aby umožnila adaptaci na různé procesy podle kategorie incidentu nebo zákazníka. Výchozím výstupem byl email zaslaný na adresu SOC týmu. Na této části jsme pracovali společně s kolegou ze školy. Hlášení incidentů je realizováno pomocí komplexního formuláře s možností připojení souborů jako důkaz o incidentu.

Mým úkolem byl naprogramovat back-end a jeho integrace s front-endem. Tedy vytvoření příslušného Modelu, Controlleru, získání dat pro formulář z databáze, ukládání připojených souborů na server a odesílání emailů.

4.3.2 Směny SOC týmu

Další funkcionalita je zobrazení seznamu analytiků a incident respondentů SOC týmu, kteří mají v daný den směnu za účelem jejich snadného kontaktování přímo z portálu. Seznam mimo kontaktu na členy obsahuje i informace o směně (od kdy do kdy trvá), časové pásmo ve kterém se nachází ale hlavně informace o aktuální dostupnosti analytika/respondenta na základě stavu komunikační aplikace Skype for Business.

Mým úkolem bylo tuto funkcionalitu implementovat. Přesněji získat data z externího softwaru na plánování směn zaměstnanců a uložit jej do databáze oddělení. Tyto data zpracovat a na jejich základě získat status z aplikace Skype for Business a informace, s možností přímého kontaktování, zobrazit na portálu.

4.3.3 Kalkulátor pro odhad ceny SIEM služeb

Poslední částí aplikace, na které jsem pracoval, je kalkulátor na výpočet odhadované ceny pro zákazníka, kterou by platil při zvolených službách. Tato kalkulace se prováděla ručně s pomocí EPS kalkulátoru od dodavatele SIEM řešení s ohledem na retenční dobu, sdíleného nebo dedikovaného hardware a jiných kritérií. Cílem tohoto řešení bylo krom interaktivního online výpočtu také umožnit využití statistik získaných z reálných dat Tieto zákazníků, což v budoucnu umožní mnohem přesnější kalkulace oproti obecným hodnotám.

Navazující součást kalkulace je online objednávka teto služby a její následné zprocesování, což je však další fáze projektu.

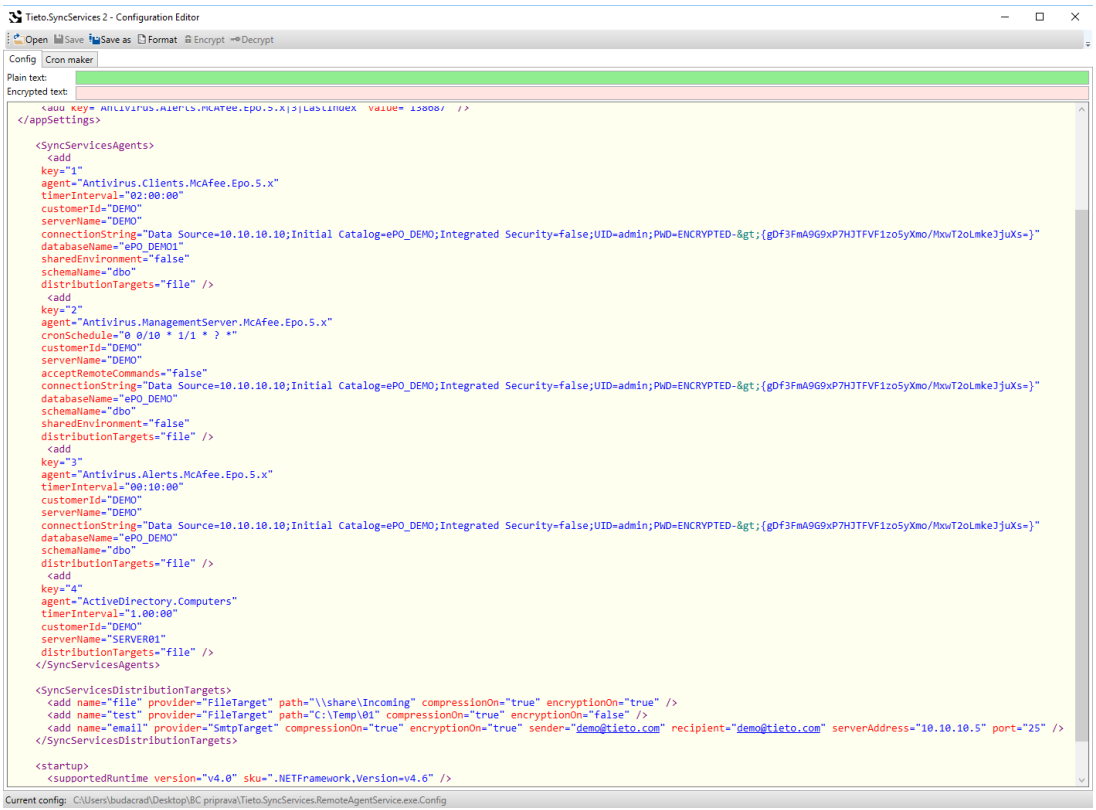
Mým úkolem bylo navrhnout logický model kalkulátoru a následně jej implementovat. Rozšířit datovou vrstvu interní aplikace o repozitář kalkulátoru, naprogramovat back-end kalkulátoru a propojit ho s formulářem, který vytvořil kolega.

5 Postup řešení zadaných úkolů

5.1 XML Editor konfiguračních souborů

Práci na mém prvním úkolu jsem začal samostudiem vzoru MVVM z internetu, kterému jsme se ve škole nevěnovali. Díky online dostupným video návodům jsem byl schopný rychle pochopit základní principy vzoru. Vytvořil jsem si testovací aplikaci, která se skládala z tlačítka a několika vstupních polích a umožnila mi si vyzkoušet nově nabyté vědomosti. Na základě zkušeností z prototypu, jsem založil nový projekt a začal pracovat na požadované aplikaci.

Práci jsem zahájil řešením GUI vrstvy napsané v jazyce XAML, protože klíčovým požadavkem bylo syntaktické zvýraznění XML kódu, jenž usnadňuje editaci konfiguračního souboru. Předlohou pro GUI mi byla starší verze podobné aplikace napsaná pomocí Windows Forms, která ovšem používala komponenty nedostupné pro WPF a bylo nutné najít vhodnou náhradu s podporou MVVM. Nakonec jsem pro tuto funkcionalitu použil AvalonEdit[4], jenž mi doporučil kolega z práce.



Obrázek 3: Screenshot aplikace s testovacím XML souborem

V dalším kroku jsem se zaměřil na základní funkce aplikace a požadavky mířené na tyto funkce, jako je otevření nebo uložení souboru, vytváření kopie originálního souboru při ukládání,

[4]Textový editor v podobě WPF komponenty[12].

nucené spuštění aplikace jako administrátor a načtení výchozího souboru při spuštění aplikace v případě, že tento soubor existuje.

V neposlední řadě jsem začal pracovat na klíčových funkcích aplikace. Nejprve bylo potřeba formátovat dokument tak, aby byl přehlednější a čitelnější. Toho jsem docílil s použitím tříd XmlWriter, XmlWriterSettings a StringBuilder. Poslední funkcí bylo uživatelsky jednoduché šifrování a dešifrování textu. K tomuto účelu jsem použil třídu Encryption což bylo jen dočasné řešení. V budoucnu byla třída nahrazena rozhraním z interní knihovny našeho oddělení.

```
private string FormatXmlDocument(XmlDocument xmlDocument){
    if (xmlDocument.DocumentElement == null)
        throw new DataException("Xml doesn't contain document part");

    var stringBuilder = new StringBuilder();
    stringBuilder.Append("<?xml version=\"1.0\" encoding=\"utf-8\"?>\r\n");
    stringBuilder.Append($"<{xmlDocument.DocumentElement.Name}>");

    var rootElement = XElement.Parse(xmlDocument.InnerXml);
    var elements = rootElement.Elements().ToArray();
    var settings = new XmlWriterSettings
    {
        OmitXmlDeclaration = true,
        Indent = true,
        NewLineOnAttributes = false,
        NewLineChars = "\n\t"
    };

    foreach (var xElement in elements)
    {
        using (var xmlWriter = XmlWriter.Create(stringBuilder, settings))
        {
            xElement.Save(xmlWriter);
        }
        stringBuilder.Replace($"<{xElement.Name}", $"\n\n\t<{xElement.Name}");
    }

    stringBuilder.Append($"\n\n</{xmlDocument.DocumentElement.Name}>");
    return stringBuilder.ToString();
}
```

Výpis 1: Funkce na formátování xml documentu v jazyce C#

Takto připravenou aplikaci jsem odeslal svému nadřízenému k revizi. Na základě zpětné vazby jsem začal na opravě nedostatků, tyto nedostatky se zčásti týkaly kódu a zčásti výsledné aplikace. Chybělo například dotazovací dialogové okno při vypnutí aplikace nebo odchytávání nepředpokládaných výjimek během chodu aplikace. Následně jsem upravil kód, aby byl přehlednější a dalo se v něm rychleji orientovat a tímto jsem zakončil práci na svém prvním úkolu.

5.2 Cron Maker

Postup řešení u mého druhého úkolu byl obdobný jako u předešlého, také jsem začal s GUI částí. Rozdíl byl v tom, že požadavkem bylo vytvoření WPF komponenty, která není sama spustitelná. Proto jsem využil prázdnou testovací aplikaci, ke které jsem připojil knihovnu a v ní vytvořil User Control[5], jenž reprezentoval mou komponentu, a začal jsem s vytvářením GUI. Oproti prvnímu úkolu bylo v GUI o poznání více prvků, ale jeho vytváření nebylo složitější a pouze zabralo více času. Pomocí jazyka XAML jsem si tedy vytvořil záložky, které jsem naplnil přepínači a vstupními poli podle předlohy.

Po dokončení GUI jsem se začal věnovat funkční části komponenty. Samostatné generování řetězce nebyl problém, stačilo pouze zjistit aktivní záložku a spustit příslušnou funkci, která pomocí vstupních hodnot z aktivní záložky vytvoří CRON řetězec.

```
private void GenerateCron(object parameter)
{
    Func<string>[] functions = {
        GenerateMinutesCron, GenerateHourlyCron,
        GenerateDailyCron, GenerateWeeklyCron,
        GenerateMonthlyCron, GenerateYearlyCron,
        GenerateOnceCron
    };
    var cronExpression = functions[_selectedTabIndex]();
    //set property to generated expresion
    CronString = cronExpression;
}
```

Výpis 2: Spuštění příslušné funkce na základě aktivní záložky

[5] Umožňuje kombinovat různé integrované ovládací prvky dohromady a zabalit je do znovu použitelného XAML kódu.

```
private string GenerateDailyCron()
{
    var cron = string.Empty;
    if (_firstdailyRadioButton)
        cron = $"0 {_selectedMinute} {_selectedHour} 1/{_everyDay} * ? *";
    else
        cron = $"0 {_selectedMinute} {_selectedHour} ? * MON-FRI *";
    return cron;
}
```

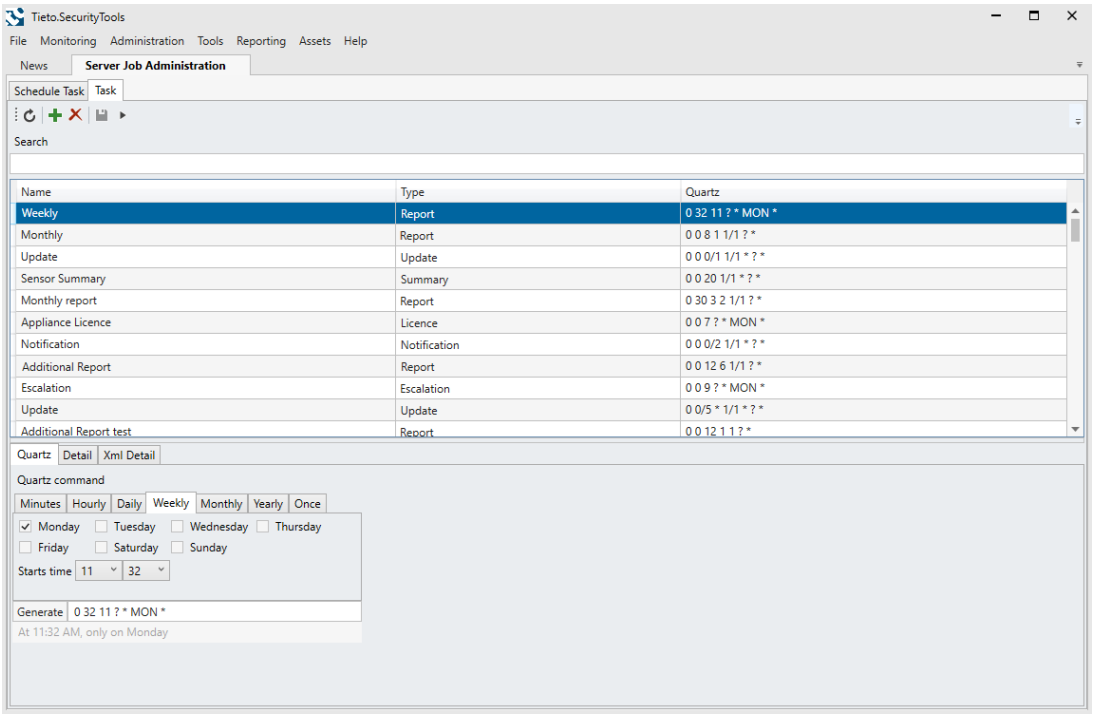
Výpis 3: Funkce na generování CRON řetězce (záložka "Daily")

V následujícím kroku jsem se zaměřil na překlad vygenerovaného řetězce do člověku srozumitelného jazyka. To slouží uživateli jako kontrola, zda řetězec reprezentuje jeho úmysl. Vlastní implementace překladu řetězců, by byla časově velmi náročná, proto jsem využil knihovnu Cron Expression Descriptor[13], která splňuje potřebnou funkčnost. Knihovna je volně dostupná jako NuGet balíček nebo webové API[6].

V poslední řadě jsem do komponenty přidal možnost zpětného rozparsování CRON řetězce. To znamená, že pokud uživatel vloží do komponenty CRON řetězec, který byl například vygenerovaný v minulosti, tak se mu nejen přeloží do srozumitelného jazyka, ale komponenta také zobrazí hodnoty z řetězce ve vstupních polích na příslušné záložce. Zobrazení na správné záložce jsem docílil pomocí regulárních výrazů, kdy jsem sledoval, jestli se v řetězci objeví klíčová část z jedné ze záložek. Následně jsem využil metody pro práci s textovými řetězci a naplnil vstupní pole či přepnul správný přepínač.

Po revizi a konzultaci s nadřízeným jsem přidal do Cron Makeru ještě jednu záložku, která umožní vytvořit CRON řetězec, jenž spustí úlohu pouze jednou ve vybraném datu. Tímto krokem jsem úspěšně dokončil svůj druhý úkol.

[6]Soubor funkcí a postupů umožňujících vytváření aplikací, které přistupují k funkcím nebo datům jiné aplikace nebo služby.



Obrázek 4: Screenshot interní aplikace s komponentou Cron Maker

5.3 SOC Portál

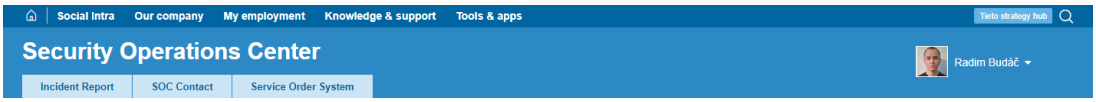
Prvním krokem při práci na SOC Portálu byla příprava aplikace na vývoj. Jelikož se jednalo o rozsáhlejší projekt a pracovalo na něm více zaměstnanců, bylo nutné vytvořit samostatný TFS projekt. Vytvořil jsem tedy novou aplikaci typu ASP.NET Core MVC a nahrál ji na TFS s nastavením verzovacího systému na Git. Následovalo nasazení aplikace na server pro testování v reálném prostředí, konfiguraci tohoto serveru zařídil kolega, a aplikace byla připravena na vývoj.

V návaznosti na zadání (viz kapitola 4.3), bylo mým prvním dílčím úkolem automatické přihlášení uživatele. Základem pro tuto funkcionalitu byla nativní podpora Windows autentizace v ASP.NET Core. Díky tomu se uživatel ihned po přístupu do aplikace transparentně přihlásí svým Windows účtem z AD, který se používá na všech klientských stanicích. Bohužel informace, které o přihlášeném uživateli získáme tímto způsobem, nejsou dostačující. Kromě uživatelského loginu bylo potřeba získat email, jméno zaměstnance a jeho fotografii, tyto informace jsou dostupné v AD Tieta. Z tohoto důvodu jsem vytvořil aplikační službu, která se pomocí loginu získaného z přihlášení připojí do AD v naší doméně. Později jsem proces přihlášení rozšířil implementací autorizační služby, která zkontroluje, zda je přihlášený uživatel členem SOC týmu a umožní mu přístup do sekcí, které nejsou pro ostatní zaměstnance Tieta přístupné.

```
//var principal = UserPrincipal.FindByIdentity(pc, IdentityType.SamAccountName,
     _domain + "\\" + currUser.Identity.Name.Remove(0, 3));

private string GetThumbnailFromPrincipial(Principal principal)
{
    var directoryEntry = (DirectoryEntry)principal.GetUnderlyingObject();
    var collection = directoryEntry.Properties["thumbnailPhoto"];

    if (!(collection.Value is byte[])) return null;

    var thumbnailInBytes = (byte[])collection.Value;
    return Convert.ToBase64String(thumbnailInBytes, Base64FormattingOptions.
        InsertLineBreaks);
}
```

Výpis 4: Funkce pro získání profilového obrázku z AD

Následovalo rozvržení sdílené webové stránky, které je společné pro všechny podstránky aplikace (pozice hlavního menu, hlavička, patička, ...). K tomu se v ASP.NET Core používá takzvaný layout, ten ve své podstatě není nic jiného než HTML soubor, který zaobalí ostatní stránky aplikace. Na základě grafického návrhu, který vytvořil kolega dedikovaný pro UI design, jsem pomocí HTML, CSS a JavaScriptu vytvořil layout stránky.



Obrázek 5: Hlavní menu webové aplikace s informacemi z AD

5.3.1 Hlášení incidentů

Hlášení incidentů byla první služba, kterou měl SOC Portál poskytovat. Na této části jsme spolupracovali s kolegou ze školy. Kolega měl na starosti front-end část (view), tedy implementaci formuláře podle grafické předlohy a klientský JavaScript kód. Mezitím co kolega pracoval na view, já začal s programováním back-endu hlášení incidentů.

Pro získávání dat z databáze, která se využívají v prvcích formuláře, jsem vycházel ze strategie Database-first, protože potřebné tabulky včetně dat již existovaly pro jinou aplikaci, se kterou tyto data sdílím. Technicky jsem využil řešení Scaffolding dostupné v Entity Framework[7], které umí vygenerovat potřebné třídy z existující databáze včetně anotací pro validaci dat. Poté jsem vytvořil asynchronní metodu na vyhledávání osob v databázi, podle jejich jména, také s využitím Entity Frameworku. Ta slouží jako našeptávač vstupního pole ve formuláři. Následovalo vytvo-

[7]Open-source ORM framework, který umožňuje snadnou práci s databází s využitím .NET objektů.

ření modelu formuláře na základě návrhu, a poté zprovoznění formuláře propojením modelu s view pomocí controlleru.

V dalším kroku bylo potřeba uložit případnou přílohu z formuláře na server. S tímto jsem neměl zkušenosti, a proto jsem se řídil návodem, který je dostupný v oficiální dokumentaci ASP.NET Core[14]. Po vyřešení ukládání příloh na server, byla na řadě hlavní podstata služby, a to výstupní akce, kdy první podporovaný výstup je odesílání emailu o incidentu. V budoucnu bude více typů výstupních akcí podle zvoleného druhu incidentu.

Emaily jsou řešené pomocí HTML šablon. Celkem se jedná o dvě šablony. Jedna pro SOC tým, kde obsah emailu reprezentuje informace vyplněné ve formuláři. Druhá pro uživatele, který formulář vyplnil, sloužící jako potvrzení že byl incident nahlášen. K samotnému odesílání emailů jsem využil službu z interní knihovny SOC oddělení.

5.3.2 Směny SOC týmu

V rámci SOC týmu spolupracují na různých vrstvách týmy z několika zemí, pro jednotný systém evidence si každý zaměstnanec SOC oddělení zapisuje směny do webové aplikace Humanity. Jedním z požadavků na SOC Portál bylo zobrazení směn analytiků a incident respondentů z oddělení k danému dni. Prvním úkolem bylo vyřešit automatickou synchronizaci dat z externího portálu do naší databáze. Humanity poskytuje pro přístup k datům REST API[15]. Na základě informací z oficiální dokumentace jsem nejprve namodeloval data a relace v aplikaci SQL data modeler pro databázi směn a zaměstnanců. Tabulku zaměstnanců už v naší databázi sice máme, ale Humanity propojuje směny a uživatele pomocí jejich ID. Rozšíření stávající tabulky o Humanity ID mi přišlo komplikované a nelogické, ne všechna oddělení v Tietu totiž Humanity používají.

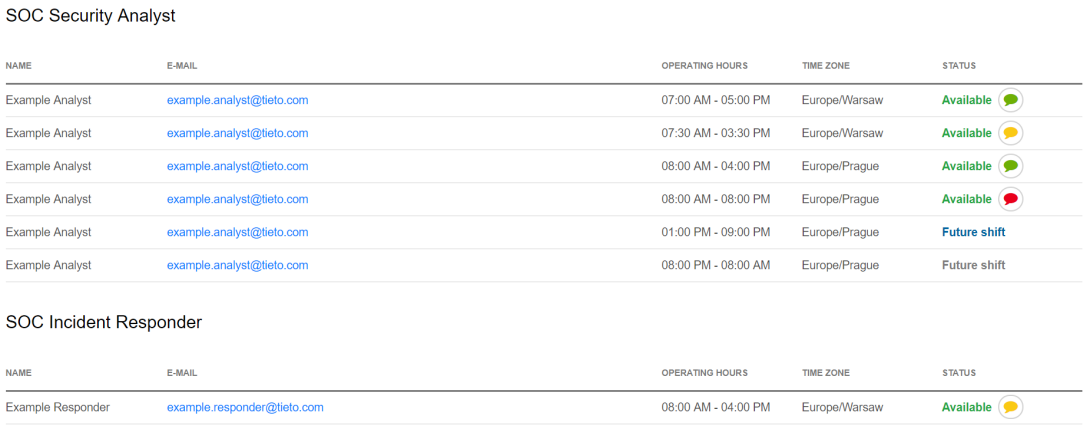
Krom obecného REST API rozhraní bylo možné využít officiálního C# SDK[8][16], které zapouzdřuje samotné API metody a umožňuje pohodlnější práci s tímto rozhraním. Díky tomuto SDK jsem byl schopný velice rychle vytvořit testovací aplikaci, která požadované data s využitím API stáhne a naplní databázi oddělení. Jelikož je směny potřeba periodicky aktualizovat, požádal jsem kolegu, aby můj kód přidal do interního plánovacího serveru jako Quartz Job a naplánoval jeho spuštění jednou denně. Zároveň jsem kontaktoval nadřízeného, aby v Humanity vytvořil servisní účet pro přístup k datům našeho oddělení, protože API vyžadovalo pro autentizaci uživatelský účet namísto obecného API klíče a mé osobní údaje byly vhodné pouze pro účel testování.

V této fázi vývoje jsem se zaměřil na SOC Portál, do kterého jsem přidal nový controller. Ten obsahoval metody pro připojení do databáze a spuštění SQL dotazu pro načtení všech analytiků a incident respondentů a informacích o jejich směnách do aplikace. Posledním požadavkem bylo zjištění statusu zaměstnance v komunikační aplikaci Skype for Business v danou chvíli.

[8]Sada nástrojů, která umožňuje vytvářet aplikace například pro určitý softwarový balíček, operační systém nebo podobnou vývojovou platformu.

K tomuto účelu jsem využil existující interní WCF službu, která na základě parametru formou pole emailových adres vrátí jako odpověď slovník <Email, Status>.

Posledním krokem bylo tyto informace vypsat na obrazovku. Pomocí jazyka HTML a CSS stylů jsem vytvořil view, kde jsou vypsáni analytici a respondenti, kteří jsou v následujících 12 hodinách dostupní dle jejich směn. Data jsou zobrazena formou tabulky včetně status ikony s odkazem na automatické otevření komunikačního okna v aplikaci Skype for Business (viz obrázek 6).


SOC Security Analyst

NAME	E-MAIL	OPERATING HOURS	TIME ZONE	STATUS
Example Analyst	example.analyst@tieto.com	07:00 AM - 05:00 PM	Europe/Warsaw	Available
Example Analyst	example.analyst@tieto.com	07:30 AM - 03:30 PM	Europe/Warsaw	Available
Example Analyst	example.analyst@tieto.com	08:00 AM - 04:00 PM	Europe/Prague	Available
Example Analyst	example.analyst@tieto.com	08:00 AM - 08:00 PM	Europe/Prague	Available
Example Analyst	example.analyst@tieto.com	01:00 PM - 09:00 PM	Europe/Prague	Future shift
Example Analyst	example.analyst@tieto.com	08:00 PM - 08:00 AM	Europe/Prague	Future shift

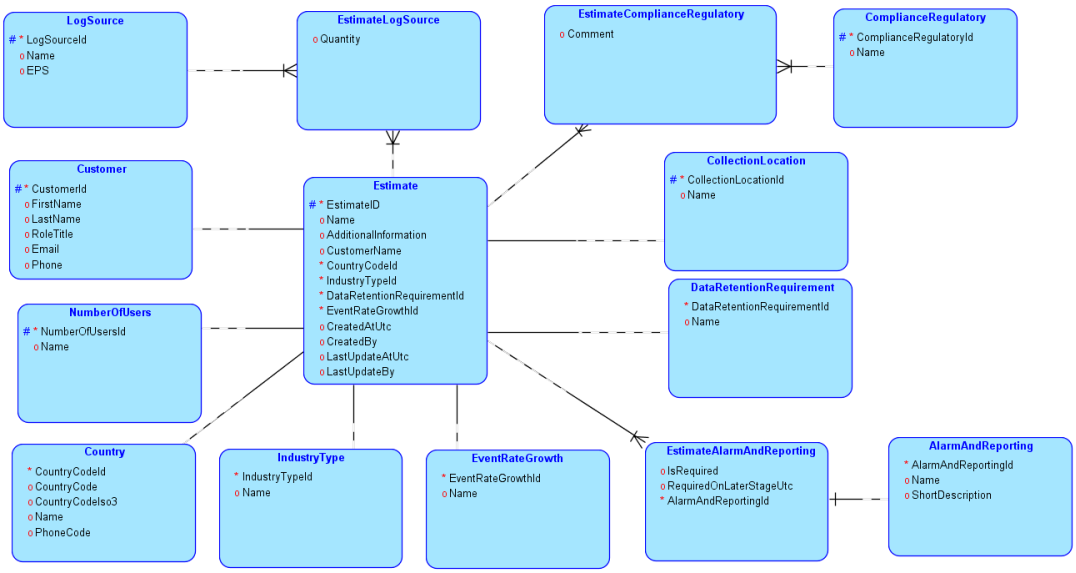
SOC Incident Responder

NAME	E-MAIL	OPERATING HOURS	TIME ZONE	STATUS
Example Responder	example.responder@tieto.com	08:00 AM - 04:00 PM	Europe/Warsaw	Available

Obrázek 6: Screenshot SOC Portálu s tabulkou směn v daném dni

5.3.3 Kalkulátor pro odhad ceny SIEM služeb

Implementační fázi kalkulátoru předcházela analýza, grafický a UX návrh, který měl na starosti kolega. Po jeho schválení jsem jej dostal jako předlohu ve formě interaktivního prototypu na portále Axure[17]. Po analýze tohoto návrhu a původního Excel souboru pro výpočet cen od dodavatele SIEM produktu, jsem vytvořil logický model kalkulátoru. Navržený model prošel revizí, požadovanými úpravami a následným schválením.



Obrázek 7: Logický model kalkulátoru

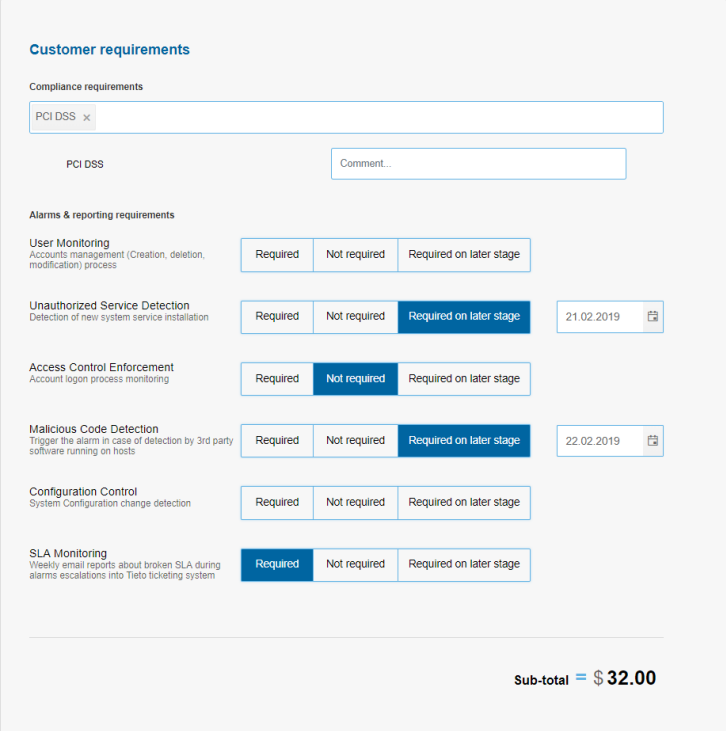
Po schválení modelu byla na řadě jeho implementace v databázi. Vytvořené tabulky jsem následně naplil daty, které jsem pro tento účel obdržel od servisního oddělení.

Dalším krokem bylo rozšíření datové vrstvy interní aplikace o repozitář kalkulátoru, který je standardně umístěn v samostatném projektu a následně sdílen pro různé interní aplikace formou NuGet balíčků. Zkušenosti s vytvářením repozitářů jsem získal ve firmě Tieto ještě před zahájením praxe. Před dokončením repozitáře jsem otestoval jeho správné fungování pomocí Unit testů.

Po dokončení datové vrstvy a pokrytím veřejných metod, jsem začal s implementací backendu kalkulátoru v SOC portálu. Jako první jsem vytvořil nový controller, který obsahoval asynchronní funkce pro získání dat z repozitáže a obsloužení uživatelských požadavků (požadavek na načtení stránky, odeslání vyplněného formuláře, načtení dříve vytvořeného odhadu). Následně zbývalo vytvořit model a propojit jej s formulářem ve view jenž vytvořil druhý kolega.

Na řadě bylo testování, zda kalkulátor správně ukládá informace z formuláře do databáze nebo správně načítá dříve vytvořené odhady. V této fázi jsem úspěšně dokončil poslední část úkolů v rámci absolvované praxe. Kalkulátor jsem předal oddělení, které na základě analýzy dat ze statistik posbíraných ze zařízení spravovaných firmou Tieto, vypočítají přesnější koeficienty

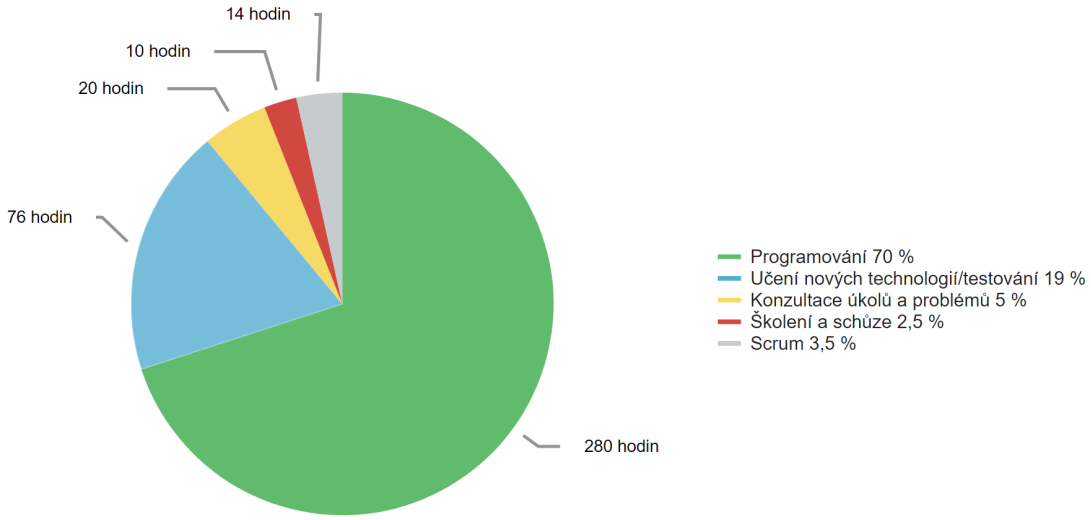
pro datový model a zaktualizují výchozí obecné hodnoty dodané výrobcem SIEM zařízení. Tím, že je tato část řešena přes databázi, tak pro tyto úpravy není nutné nijak zasahovat do aplikačního kódu samotného kalkulátoru.



Obrázek 8: Screenshot SOC Portálu s částí formuláře kalkulátoru

6 Časové rozvržení

Následující graf znázorňuje čas strávený na praxi v průběhu 50 dní při běžné pracovní době 8 hodin za den, rozdělený do patřičných skupin. Graf je pouze odhadem a od skutečného časového rozložení se může mírně lišit.



Obrázek 9: Graf časového rozložení během praxe

Tabulka níže zobrazuje časovou náročnost jednotlivých úkolů. Počet dnů v tabulce znázorňuje období, ve kterém jsem na úkolu pracoval, tedy od zadání úkolu po jeho splnění.

Tabulka 2: Časová náročnost úkolů

Úkol	Počet dnů
XML Editor konfiguračních souborů	10
Cron Maker	10
SOC Portál	
Hlášení incidentů	10
Směny SOC týmu	5
Kalkulátor pro odhad ceny služeb	15

7 Závěr

7.1 Uplatněné znalosti

Při absolvování odborné praxe jsem využil znalosti z nejednoho předmětu z mého studia. Jako nejdůležitější považuji znalost jazyka C#, se kterým jsem během praxe programoval. Další důležité vědomosti, které jsem měl možnost uplatnit, byly z oblasti databází a to návrhu logického modelu, jeho pozdější implementace nebo vytváření netriviálních SQL dotazů. Dále jsem měl možnost uplatnit své znalosti webových technologií.

Velkou pomocí mi byl předmět Architektura technologie .NET, vědomosti z tohoto předmětu jsem využil při mém posledním úkolu.

7.2 Scházející a nabyté znalosti

Mezi znalosti, které mi během odborné praxe scházely, patří například již zmíněný vzor MVVM, se kterým jsem se ve škole nesetkal. Také používání distribuovaných systému na správu verzí, jako je Git, jsem měl možnost vyzkoušet až na odborné praxi. Naučil jsem se vytvářet WPF komponenty, rozšířil jsem své znalosti v oblasti webových technologií, zdokonalil jsem se v C# jazyce a zlepšil svou práci v týmu.

Všechny nastalé problémy a překážky nebyl problém překonat, a to díky samostudiu, ale hlavně díky pomoci kolegů.

7.3 Dosažené výsledky

První dva úkoly byli spíše individuální. Výsledkem prvního úkolu je aplikace, která usnadní práci při konfiguraci Windows služeb. Druhý úkol vedl k vytvoření komponenty, jenž usnadní práci, ale hlavně ušetří čas kolegům. Od kolegů mi přišla velice pozitivní zpětná vazba, díky tomu jsem věděl, že práce na komponentě měla smysl a byla přínosná. Při práci na třetím úkolu jsem byl u zrodu větší interní aplikace a podílel se na vývoji prvotních funkcionalit této víceúčelové aplikace.

7.4 Zhodnocení praxe

Absolvování individuální odborné praxe hodnotím velmi kladně, znalosti a hlavně zkušenosti nabyté při praxi jsou pro mě neocenitelné. Vyzkoušet si, jak probíhá pracovní proces v tak velké firmě, bylo velice přínosné a důležité do budoucna. Práce nebyla jednotvárná a bavila mě, svého rozhodnutí o způsobu provedení bakalářské práce nelituji.

Literatura

[1] Tieto historie [online]. [cit. 2019-2-21].
Dostupné z: https://campaigns.tieto.com/tieto50/

[2] O společnosti Tieto [online]. [cit. 2019-2-21].
Dostupné z: https://www.tieto.com/en/about-us/our-company/

[3] O pobočce Tieto Czech [online]. [cit. 2019-2-21].
Dostupné z: https://www.tieto.com/cz/about-us/our-company/

[4] Visual Studio [online]. [cit. 2019-2-21].
Dostupné z: https://visualstudio.microsoft.com/

[5] SQL Server Management Studio [online]. [cit. 2019-2-21].
Dostupné z: https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-2017

[6] Jira Software [online]. [cit. 2019-2-21].
Dostupné z: https://www.atlassian.com/software/jira

[7] Humanity [online]. [cit. 2019-2-21].
Dostupné z: https://www.humanity.com/

[8] Team Foundation Server [online]. [cit. 2019-2-21].
Dostupné z: https://visualstudio.microsoft.com/tfs/

[9] ASP.NET Core [online]. [cit. 2019-2-21].
Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2

[10] Cron Maker [online]. [cit. 2018-2-21].
Dostupné z: http://www.cronmaker.com/

[11] Quartz Enterprise Scheduler .NET [online]. [cit. 2019-3-5].
Dostupné z: https://www.quartz-scheduler.net

[12] AvalonEdit [online]. [cit. 2019-3-7].
Dostupné z: http://avalonedit.net

[13] Cron Expression Descriptor [online]. [cit. 2019-3-14].
Dostupné z: https://cronexpressiondescriptor.azurewebsites.net/

[14] Nahrávání souborů v ASP.NET Core [online]. [cit. 2019-3-19].
Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/mvc/models/file-uploads

[15] Humanity API [online]. [cit. 2019-3-19].
Dostupné z: https://platform.humanity.com/

[16] Humanity API SDK [online]. [cit. 2019-3-19].
Dostupné z: https://github.com/shiftplanning/cs-sdk

[17] Axure [online]. [cit. 2019-3-19].
Dostupné z: https://www.axure.com/